2008 Eighth IEEE International Conference on Data Mining

A Practical Approach to Classify Evolving Data Streams: Training with Limited Amount of Labeled Data

Mohammad M. Masud[†] mehedy@utdallas.edu
Jing Gao[‡] jinggao3@uiuc.edu
Latifur Khan[†] lkhan@utdallas.edu
Jiawei Han[‡] hanj@cs.uiuc.edu
Bhavani Thuraisingham[†] bhavani.thuraisingham@utdallas.edu

[†]Department of Computer Science, University of Texas at Dallas
[‡]Department of Computer Science, University of Illinois at Urbana-Champaign

Abstract

Recent approaches in classifying evolving data streams are based on supervised learning algorithms, which can be trained with labeled data only. Manual labeling of data is both costly and time consuming. Therefore, in a real streaming environment, where huge volumes of data appear at a high speed, labeled data may be very scarce. Thus, only a limited amount of training data may be available for building the classification models, leading to poorly trained classifiers. We apply a novel technique to overcome this problem by building a classification model from a training set having both unlabeled and a small amount of labeled instances. This model is built as micro-clusters using semi-supervised clustering technique and classification is performed with κ-nearest neighbor algorithm. An ensemble of these models is used to classify the unlabeled data. Empirical evaluation on both synthetic data and real botnet traffic reveals that our approach, using only a small amount of labeled data for training, outperforms state-of-the-art stream classification algorithms that use twenty times more labeled data than our approach.

1 Introduction

Stream data classification is a challenging problem because of two important properties: its infinite length and evolving nature. Data streams may evolve in several ways: the prior probability distribution $p(c)$ of a class c may change, or the posterior probability distribution $p(c|x)$ of the class may change, or both the prior and posterior probabilities may change. In either case, the challenge is to build a classification model that is consistent with the current concept. Traditional learning algorithms that require several passes on the training data, cannot be directly applied to the streaming environment, because the number of training examples would be infinite. To solve this problem, ensemble classification techniques have been proposed.

Ensemble approaches have the advantage that they can be updated efficiently, and they can be easily made to adopt the changes in the stream. Several ensemble approaches have been devised for classification of evolving data streams [7, 10]. The general technique practiced by these approaches is that the data stream is divided into equal-sized chunks. Each of these chunks is used to train a classifier. An ensemble of L such classifiers are used to test unlabeled data. However, these ensemble approaches are based on supervised learning algorithms, and can be trained only with labeled data. But in practice, labeled data in streaming environment are rare.

Manual labeling of data is usually costly and time consuming. So, in an streaming environment, where data appear at a high speed, it may not be possible to manually label all the data as soon as they arrive. Thus, in practice, only a small fraction of each data chunk is likely to be labeled, leaving a major portion of the chunk as unlabeled. So, a very limited amount of training data will be available for the supervised learning algorithms. Considering this difficulty, we propose an algorithm that can handle "partially labeled" training data in a streaming environment. By "partially labeled" we mean only a fraction (e.g. 5%) of the training instances are labeled, and by "completely labeled" we mean all (100%) the training instances are labeled. Our approach is capable of producing the same (or even better) results with *partially labeled* training data compared to other approaches that use *completely labeled* training data having twenty times more labeled data than our approach.

Naturally, stream data could be stored in buffer and pro-

1550-4786/08 $25.00 © 2008 IEEE
DOI 10.1109/ICDM.2008.152

cessed when the buffer is full, so we divide the stream data into equal sized chunks. We train a classification model from each chunk. We propose a semi-supervised clustering algorithm to create K clusters from the partially labeled training data. A summary of the statistics of the instances belonging to each cluster is saved as a "micro-cluster". These micro-clusters serve as a classification model. To classify a test instance using this model, we apply the κ-nearest neighbor (κ-NN) algorithm to find the Q nearest micro-clusters from the instance and select the class that has the highest frequency of labeled data in these Q clusters. In order to cope with the stream evolution, we keep an ensemble of L such models. Whenever a new model is built from a new data chunk, we update the ensemble by choosing the best L models from the L+1 models (previous L models and the new model), based on their individual accuracies on the labeled training data of the new data chunk. Besides, we refine the existing models in the ensemble whenever a new class of data evolves in the stream.

It should be noted that when a new data point appears in the stream, it may not be labeled immediately. We defer the ensemble updating process until some data points in the latest data chunk have been labeled, but we keep classifying new unlabeled data using the current ensemble. For example, consider the online credit-card fraud detection problem. When a new credit-card transaction takes place, its class ($\{$fraud,authentic$\}$) is predicted using the current ensemble. Suppose a 'fraud' transaction has been mis-classified as 'authentic'. When the customer receives the bank statement, he will identify this error and report to the authority. In this way, the actual labels of the data points will be obtained, and the ensemble will be updated accordingly.

We have several contributions. First, we propose an efficient semi-supervised clustering algorithm based on cluster-impurity measure. Second, we apply our technique to classify evolving data streams. To our knowledge, there are no stream data classification algorithms that apply semi-supervised clustering. Third, we provide a solution to the more practical situation of stream classification when labeled data are scarce. We show that our approach can achieve better classification accuracy than other stream classification approaches, utilizing only a fraction (e.g. 5%) of the labeled instances used in those approaches. Finally, we apply our technique to detect botnet traffic, and obtain 98% classification accuracy on average. We believe that the proposed method provides a promising, powerful, and practical technique to the stream classification problem in general.

The rest of the paper is organized as follows: section 2 discusses related work, section 3 describes the semi-supervised clustering technique, section 4 discusses the ensemble classification with micro-clusters, section 5 discusses the experiments and evaluation of our approach, and section 6 concludes with directions to future works.

2 Related work

Our work is related to both stream classification and semi-supervised clustering techniques. We briefly discuss both of them.

Semi-supervised clustering techniques utilize a small amount of knowledge available in the form of pairwise constraints (*must-link, cannot-link*), or class labels of the data points. Recent approaches for semi-supervised clustering incorporated pairwise constraints on top of the unsupervised K-means clustering algorithm and formulated a constraint-based K-means clustering problem [2, 9], which was solved with an Expectation-Maximization (E-M) framework. Our approach is different from these approaches because rather than using pair-wise constraints, we utilize a cluster-impurity measure based on the limited labeled data contained in each cluster. If pair-wise constraints are used, then the running time per E-M step is quadratic in total number of labeled points, whereas the running time is linear if impurity measures are used. So, the impurity measures are more realistic in classifying a high-speed stream data.

There have been many works in stream data classification. There are two main approaches - single model classification, and ensemble classification. Single model classification techniques incrementally update their model with new data to cope with the evolution of the stream [4, 5]. These techniques usually require complex operations to modify the internal structure of the model and may perform poorly if there is concept-drift in the stream. To solve these problems, several ensemble techniques for stream data mining have been proposed [7, 10]. These ensemble approaches have the advantage that they can be more efficiently built than updating a single model and they observe higher accuracy than their single model counterpart [8].

Our approach is also an ensemble approach, but it is different from other ensemble approaches in two aspects. First, previous ensemble-based techniques use the underlying learning algorithm (such as decision tree, Naive Bayes, etc.) as a black-box and concentrate only on building an efficient ensemble. But we concentrate on the learning algorithm itself, and try to construct efficient classification models in an evolving scenario. In this light, our work is more closely related with the work of Aggarwal et al [1]. Secondly, previous techniques (including [1] require *completely labeled* training data. But in practice, a very limited amount of labeled data may be available in the stream, leading to poorly trained classification models. So our approach is more realistic in a stream environment. Our model is also different from Aggarwal et al. [1] in one more aspect: Aggarwal et al. apply horizon-fitting to classify evolving data streams, whereas we use a fixed-sized ensemble of classifiers, which requires less memory since we do not need to store snapshots.

3 Impurity-based clustering with small number of labeled data

In the semi-supervised clustering problem, we are given m data points $\mathcal{X} = \{\boldsymbol{x_1}, \boldsymbol{x_2}, ..., \boldsymbol{x_m}\}$, and their corresponding class labels, $\mathcal{Y} = \{\boldsymbol{y_1}, \boldsymbol{y_2}, ..., \boldsymbol{y_m}\}$, $y_j \in \{\phi, 1, ..., C\}$ where C is the total number of classes. If a data point $x_j \in \mathcal{X}$ has $y_j = \phi$, then it is unlabeled. We are to create K clusters, maintaining the constraint that all *labeled* points in the same cluster have the same class label. Given a limited amount of labeled data, the goal of impurity-based clustering is to create K clusters by minimizing the intra-cluster dispersion (same as unsupervised K-means) and at the same time minimizing the impurity of each cluster. We will refer to this problem as K-means with Minimization of Cluster Impurity (MCI-Kmeans). A cluster is completely pure if it contains labeled data points from only one class (along with some unlabeled data). Thus, the objective function should penalize each cluster for being impure. The general form of the objective function is as follows:

$$\mathcal{O}_{MCIKmeans} = \sum_{i=1}^{K} \sum_{\boldsymbol{x} \in \mathcal{X}_i} ||\boldsymbol{x} - \boldsymbol{u_i}||^2 + \sum_{i=1}^{K} \mathcal{W}_i * Imp_i \quad (1)$$

where $\mathcal{W}_i$ is the weight associated with cluster i and Imp_i is the impurity of cluster i. In order to ensure that both the intra-cluster dispersion and cluster impurity are given the same importance, the weight associated with each cluster should be adjusted properly. Besides, we would want to penalize each data point that contributes to the impurity of the cluster (i.e., the labeled points). So, the weight associated with each cluster is chosen to be

$$\mathcal{W}_i = |\mathcal{L}_i| * \bar{D}_{\mathcal{L}_i} \quad (2)$$

where $\mathcal{L}_i$ is the set of all labeled data points in Cluster i and $\bar{D}_{L_i}$ is the average dispersion from each of these labeled points to the cluster centroid. Thus, each labeled point has a contribution to the total penalty, which is equal to the cluster impurity multiplied by the average dispersion of the labeled points from the centroid. We observe that equation (2) is equivalent to the sum of dispersions of all labeled points from the cluster centroid, i.e., $\mathcal{W}_i = \sum_{\boldsymbol{x} \in \mathcal{L}_i} ||\boldsymbol{x} - \boldsymbol{u_i}||^2$. Substituting this value of $\mathcal{W}_i$ in (1) we obtain:

$$\mathcal{O}_{MCIKmeans} = \sum_{i=1}^{K} \sum_{\boldsymbol{x} \in \mathcal{X}_i} ||\boldsymbol{x} - \boldsymbol{u_i}||^2 + \sum_{i=1}^{K} \sum_{\boldsymbol{x} \in \mathcal{L}_i} ||\boldsymbol{x} - \boldsymbol{u_i}||^2 * Imp_i$$

$$= \sum_{i=1}^{K} (\sum_{\boldsymbol{x} \in \mathcal{X}_i} ||\boldsymbol{x} - \boldsymbol{u_i}||^2 + \sum_{\boldsymbol{x} \in \mathcal{L}_i} ||\boldsymbol{x} - \boldsymbol{u_i}||^2 * Imp_i) \quad (3)$$

Impurity measures: Equation (3) should be applicable to any impurity measure in general. We use the following impurity measure: $Imp_i = \mathcal{ADC}_i * Ent_i$, where $\mathcal{ADC}_i$ is the "aggregated dissimilarity count" of cluster i and Ent_i is the entropy of cluster i. In order to understand $\mathcal{ADC}_i$, we first need to define "Dissimilarity count".

Definition 1 (Dissimilarity count) *Dissimilarity count $\mathcal{DC}_i(\boldsymbol{x}, y)$ of a data point $\boldsymbol{x}$ in cluster i having class label y is the total number of labeled points in that cluster belonging to classes other than y. If $\boldsymbol{x}$ is unlabeled (i.e., $y = \phi$), then $\mathcal{DC}_i(\boldsymbol{x}, y)$ is zero.*

In other words, $\mathcal{DC}_i(\boldsymbol{x}, y) = 0$, if $\boldsymbol{x}$ is unlabeled, and $\mathcal{DC}_i(\boldsymbol{x}, y) = |\mathcal{L}_i| - |\mathcal{L}_i(c)|$, if $\boldsymbol{x}$ is labeled and its label $y = c$, where $\mathcal{L}_i(c)$ is the set of labeled points in cluster i belonging to class c. Note that $\mathcal{DC}_i(\boldsymbol{x}, y)$ can be computed in constant time, if we keep an integer vector to store the counts $|\mathcal{L}_i(c)|, c \in \{1, .., C\}$. "Aggregated dissimilarity count" or $\mathcal{ADC}_i$ is the sum of the dissimilarity counts of all the points in cluster i: $\mathcal{ADC}_i = \sum_{x \in \mathcal{L}_i} DC_i(\boldsymbol{x}, y)$. Entropy of a cluster i is computed as: $Ent_i = \sum_{c=1}^{C} (-p_c^i * log(p_c^i))$, where p_c^i is the prior probability of class c, i.e., $p_c^i = \frac{|\mathcal{L}_i(c)|}{|\mathcal{L}_i|}$.

The use of Ent_i in the objective function ensures that clusters with higher entropy get higher penalties. However, if only Ent_i had been used as the impurity measure, then each labeled point in the same cluster would have received the same penalty. But we would like to favor the labeled points belonging to the majority class in a cluster, and disfavor the points belonging to the minority classes. Doing so would force more labeled points of the majority class to be moved into the cluster, and more labeled points of the minority classes to be moved out of the cluster, making the clusters purer. This is ensured by introducing $\mathcal{ADC}_i$ to the equation. We call the combination of $\mathcal{ADC}_i$ and Ent_i as "compound impurity measure" since it can be shown that $\mathcal{ADC}_i$ is proportional to the "gini index" of cluster i:

$$\mathcal{ADC}_i = \sum_{c=1}^{C} (|\mathcal{L}_i(c)|)(|\mathcal{L}_i| - |\mathcal{L}_i(c)|) = (|\mathcal{L}_i|)^2 \sum_{c=1}^{C} (p_c^i)(1 - p_c^i)$$

$$= (|\mathcal{L}_i|)^2 (1 - \sum_{c=1}^{C} (p_c^i)^2) = (|\mathcal{L}_i|)^2 * Gini_i$$

where $Gini_i$ is the gini index of cluster i.

The problem of minimizing equation (3) is an *incomplete-data problem* because the cluster labels and the centroids are all unknown. The common solution to this problem is to apply E-M [3]. The E-M algorithm consists of three basic steps: initialization, E-step and M-step. The technical details of these steps can be found in [6].

4 Micro-clustering and ensemble training

After creating K clusters using the semi-supervised algorithm, we extract and save summary of the statistics of the data points in each cluster as a "micro-cluster" and discard the raw data points. We will refer to the K micro-clusters built from a data chunk as a classification model, since we use these micro-clusters to classify unlabeled data. We keep an ensemble of L such models.

4.1 Storing the cluster summary information as micro-clusters

Each model $M^i \in M$ contains K micro-clusters $\{M_1^i, ..., M_K^i\}$, where each micro-cluster M_j^i is a summary of

the statistics of the data points $\mathcal{X}_j^i = \{\boldsymbol{x}_{j_1}^i, ...\boldsymbol{x}_{j_N}^i\}$ belonging to that cluster. The summary contains the following statistics: i) N: the total number of points; ii) Lt: the total number of labeled points; iii) $\{Lp[c]\}_{c=1}^{C}$: a vector containing the total number of labeled points belonging to each class. iv) $\boldsymbol{u}$: the centroid of the cluster; v) $\{Sum[r]\}_{r=1}^{d}$: a vector containing the sum of each dimension of the data points in the cluster, where $Sum[r]$ contains the sum of the values of the r^{th} dimension. This vector is required to re-compute the cluster centroid after merging two micro-clusters.

4.2 Updating the ensemble

Every time a new data chunk D_n appears, we train a new model M^n from D_n and update the ensemble by choosing the best L models from the existing L+1 models ($M \cup \{M^n\}$). Algorithm 1 sketches this updating process.

Algorithm 1 Ensemble-Update

Input: $\mathcal{X}^n, \mathcal{Y}^n$: training data points and class labels associated with some of these points in chunk D_n
$\mathcal{Z}^n$: test data points in chunk D_n
K: number of clusters to be created
M: current ensemble of L models $\{M^1, ..., M^L\}$
Output: Updated ensemble M

1: Obtain K clusters $\{\mathcal{X}_1^n, ..., \mathcal{X}_K^n\}$ using E-M algorithm. and compute their summary of statistics $\{M_1^n, ..., M_K^n\}$
2: **if** no cluster $M_j^i \in M$ contains some class c that is seen in the new model M^n **then**
3: Refine-Ensemble(M, M^n)
4: **end if**
5: Test each model $M_j^i \in M$ and M^n on the labeled data of $\mathcal{X}^n$ and obtain its accuracy
6: $M \leftarrow$ Best L models in $M \cup \{M^n\}$ based on accuracy.
7: Predict the class labels of data points in $\mathcal{Z}^n$ with M.

Description of the algorithm "Ensemble-Update": Assuming that the new data chunk D_n has *some* labeled data, we first randomly divide it into two subsets; $\mathcal{X}^n$: the training set and $\mathcal{Z}^n$: the test set. We include all the labeled instances and a few unlabeled instances from D_n in the training set. The rest of the unlabeled instances in D_n are included in the test set. We create K clusters using $\mathcal{X}^n$ with the clustering technique described in section 3. We then extract the summary of statistics from each cluster $\mathcal{X}_j^n$ and store it as a micro-cluster M_j^n (line 1). We handle a special case in lines (2-4) that deals with the evolving data streams. It is possible that in the new data chunk, suddenly a new class has appeared that never appeared in the stream before. Or it may happen that a class has appeared, which has not been in the stream for a long time. In either case, the class is unknown to the existing ensemble of models M. So, we refine the models in M so that they can correctly classify the instances belonging to that class. This refinement process will be explained shortly. Since we have L+1 models now, one of them must be discarded. This is done by testing the accuracy of each of these models on the labeled data points in the training data $\mathcal{X}^n$, and removing the worst of them (lines 5-6). Finally, we predict the classes of the test data $\mathcal{Z}^n$ with the new ensemble M (line 7).

Ensemble refinement: The ensemble M is refined using the newly built model M^n. The refinement procedure first looks into each micro-cluster M_j^n of the model M^n. If any micro-cluster has some labeled data and majority of the labeled data are in class $\hat{c}$, but no model in the ensemble M has any micro-cluster containing labeled data of class $\hat{c}$, then we do the following: for each model $M^i \in M$, we inject the micro-cluster M_j^n in M^i with some probability, called the *probability of injection*, or ρ. To inject a micro-cluster, we first merge two nearest micro-clusters in M^i having the same majority class. Then we add the new micro-cluster M_j^n to M^i. This ensures that total number of micro-clusters in the model remains constant.

The reasoning behind this refinement is as follows. Since no model in ensemble M has knowledge of the class $\hat{c}$, the models will certainly miss-classify any data belonging to the class. By injecting micro-clusters of the class $\hat{c}$, we introduce some data from this class into the models, which reduces their miss-classification rate. It is obvious that for higher values of ρ, more training instances will be provided to a model, which will probably induce more error reduction. So, when $\rho = 1$, we will probably have maximum reduction in prediction error for a *single model*. However, if the same set of micro-clusters are injected in all the models, then the correlation among them may increase, resulting in reduced prediction accuracy of the *ensemble* [8]. Lemma 1 states that the ensemble error is the lowest when $\rho = 0$.

Lemma 1 *Let $\mathcal{E}_M$ be the added error of the ensemble M when $\rho \geq 0$ and $\mathcal{E}_M^0$ is the added error of the ensemble M when $\rho = 0$. Then $\mathcal{E}_M \geq \mathcal{E}_M^0$ for any $\rho \geq 0$.*

Proof: See [6]. □

So, in summary, if we increase ρ, single model error decreases but the ensemble error increases. So, the net effect is that when ρ is initially increased from zero, the overall error keeps decreasing upto a certain point. After that point, increasing ρ hurts performance (i.e., the total error starts increasing) due to increased correlation among the models. This trade-off is also discussed in our experimental results (section 5.3). So, we have to choose a value of ρ that can minimize the overall error. In our experiments, the best value was found to be within 0.5-0.75.

4.3 Ensemble classification using κ-nearest neighbor

In order to classify an unlabeled data point $\boldsymbol{x}$ with a model M^i, we perform the following steps: i) find the Q-nearest *labeled* micro-clusters from $\boldsymbol{x}$ in M^i, by computing the distance between the point and the centroids of the micro-clusters. A micro-cluster is assumed to be *labeled* if

it has at least one labeled data point. ii) select the class with the highest "cumulative normalized-frequency (CNFrq)" in these Q clusters as the predicted class of $\boldsymbol{x}$. The "normalized frequency" of a class c in a micro-cluster is the number of instances of class c divided by the total number of labeled instances in that micro-cluster. CNFrq of a class c is the sum of the normalized frequencies of class c in all the Q clusters. In order to classify $\boldsymbol{x}$ with the *ensemble* M, we perform the following steps: i) find the Q-nearest *labeled* micro-clusters from $\boldsymbol{x}$ in each model $M^i \in M$, ii) select the class with the highest CNFrq in these $L * Q$ clusters as the predicted class.

5 Experiments

In this section we discuss the data sets used in the experiments, the system setup, the results, and analysis.

5.1 Datasets and experimental setup

We apply our technique on real botnet dataset generated in a controlled environment, and also on synthetic datasets. Details of these datasets are discussed in [6].

Each dataset is divided into two equal subsets: training and testing, such that every training instance is followed by a test instance. Our algorithm will be mentioned henceforth as "SmSCluster", which is the acronym for Semi-supervised Stream Clustering. Parameter settings of SmSCluster are as follows, unless mentioned otherwise: K (number of micro-clusters) = 50; Q (number of nearest neighbors for the κ-NN classification) = 1; ρ (probability of injection) = 0.75; Chunk-size = 1,600 records for botnet dataset, and 1,000 records for synthetic dataset; L (ensemble size) = 8; P (Percentage of labeled points): 5% in all datasets, meaning only 5% (randomly selected) of the training data are assumed to have labels;

We compare our algorithm with that of Aggarwal et al [1]. We will refer to this approach as "On Demand Stream". For the On Demand Stream, we use all the default values of its parameters. We use the same set of training and test data for both On Demand Stream and SmSCluster with the only difference that in SmSCluster, only 5% data in the training set have labels, but in On Demand Stream, 100% data in the training set have labels. So, if there are *100 data points in a training set, then On Demand Stream has 100 labeled training data points, but SmSCluster has only 5 of them labeled and 95 of them unlabeled*. Also, for a fair comparison, the chunk-size of SmSCluster is made equal to the buffer size of On Demand Stream. We run our own implementation of the On Demand Stream and report the results.

5.2 Comparison with baseline methods

Figure 1(a) shows the accuracy of SmSCluster and On Demand Stream (for "stream_speed"=80, "buffer_size"=1,600, and k_{fit}=80) on the botnet data. We also obtain similar results for other values of "stream_speed" and "buffer_size" for On Demand Stream. The X-axis represents stream in time units and the Y-axis represents accuracy. Here each time unit is equal to 80 data points. For example, the left bar at time unit 120 (X=120) shows the accuracy of of SmSCluster at that time, which is 98%. The right bar at the same time unit shows the accuracy of On Demand Stream, which is 94%. SmSCluster has 4% or better accuracy than On Demand Stream in all the five time-stamps shown in the chart. Figure 1(b) shows

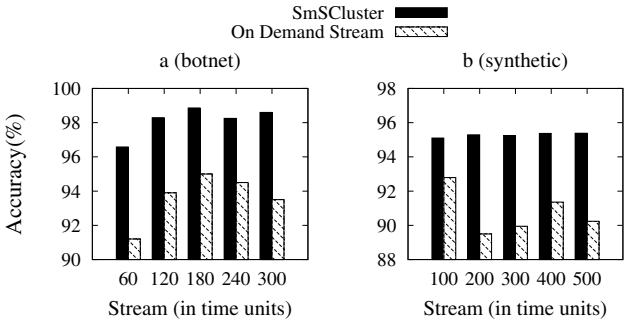

Figure 1. Accuracy comparison on (a) botnet data, and (b) synthetic data

the accuracies of SmSCluster and On Demand Stream (for "stream_speed"=200, "buffer_size"=1,000 , and k_{fit}=50) on synthetic data (100K, C10, D40). We also obtain similar results for other values of "stream_speed" and "buffer_size" for On Demand Stream. SmSCluster has 4% or better accuracy than On Demand Stream in all time units except at time 100, when the difference is 2.3%.

From the above results, we can conclude that SmSCluster outperforms On Demand Stream in all datasets. There are two main reasons behind this. First, SmSCluster considers both the dispersion and impurity measures in building clusters, but On Demand Stream considers only purity, since it applies supervised K-means algorithm. Besides, SmSCluster uses proportionate initialization, so that more clusters are formed for the larger classes (i.e., classes having more instances). But On Demand Stream builds equal number of clusters for each class, so clusters belonging to larger classes may be bigger (and more sparse). Thus, the clusters of SmSCluster are likely to be more compact than those of the On Demand Stream. As a result, the κ-nearest neighbor classification gives better prediction accuracy in SmSCluster. Second, SmSCluster applies ensemble classification, rather than the "horizon fitting" technique used in On Demand Stream. Horizon fitting selects a horizon of training data from the stream that corresponds to a variable-length window of the most recent (contiguous) data chunks. It may be possible that one or more chunks in that window have been outdated, resulting in a less accurate classification model. This is because the set of training data that is the best representative of the current concept are not necessarily contiguous. But SmSCluster always keeps the best

training data (or models) that are not necessarily contiguous. So, the ensemble approach is more flexible in retaining the most up-to-date set of training data, resulting in a more accurate classification model.

5.3 Running times and sensitivity to parameters

The processing speed of SmSCluster for botnet data is 4,000 instances per second, and for synthetic data (300K, C10, D20) 2,500 instances per second, including training and testing instances. Speed is faster for the botnet data since it has only 2 classes, as opposed to 10 classes for the synthetic data. Besides, experimental results show that [6] running time of SmSCluster scales linearly to higher dimensionality and class labels.

All the following results are obtained from the synthetic data (300K, C10, D20), but these are the general trends in any dataset. Figure 2(a) shows how the classification accuracy varies for SmSCluster with the number of micro-clusters (K), and the number of nearest neighbors (Q) for the κ-NN algorithm. We observe that higher values of K lead to better classification accuracies. This may happen because when K is larger, smaller and more compact clusters are formed, leading to a finer-grained classification model for the κ-NN algorithm. However, there is no significant improvement after K=50. We also observe the effect of Q from this chart. It is evident that Q=1 has the highest accuracy, meaning, we need to apply only 1-nearest neighbor. This is true for any value of K. Figure 2(b) shows how the classification accuracy varies for SmSCluster with percentage of labeled data (P) in the training set and the number of micro-clusters (K). We see that the accuracy increases with increasing number of labeled data in the training set. This is desirable, because more labeled data means better guidance for clustering, leading to reduced error. However, after a certain point (20%), there is no real improvement. This is because, probably this amount of labeled data is sufficient for the model. The parameters ρ (injection probability) and L (ensemble size) also have effects on accuracy. We observe that increasing ρ (injection probability) up to 0.5 increases the overall accuracy. After that, the accuracy drops in general. This result follows from our analysis discussed in section 4.2. We achieve the highest accuracy for ensemble size (L)=8. Further increasing the ensemble size does not improve the performance. This is possible if the dataset evolves continuously, resulting in some out-dated models in a larger ensemble.

6 Conclusion

We address a more realistic problem of stream mining: training with a limited amount of labeled data. Our technique is a more practical approach to the stream classification problem since it requires a fewer amount of labeled data, saving much time and cost that would be otherwise required to manually label the data. Previous approaches for stream classification did not address this vital problem. We propose and implement a semi-supervised clustering based stream classification algorithm to solve this limited labeled-data problem. We tested our technique on synthetically generated dataset, and real botnet dataset, and got better classification accuracies than other stream classification techniques. In future, we would like to incorporate feature-weighting and distance-learning in the semi-supervised clustering.

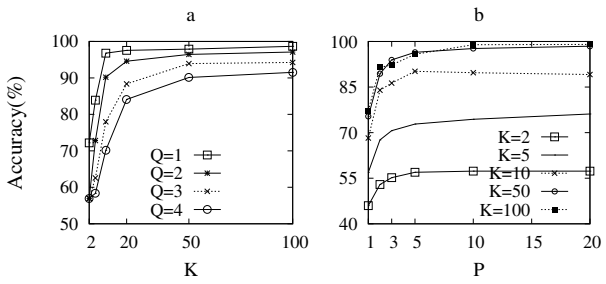

Figure 2. Sensitivity to parameters P, K, Q

References

[1] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for on-demand classification of evolving data streams. *IEEE Transactions on Knowledge and Data Engineering*, 18(5):577–589, 2006.

[2] S. Basu, M. Bilenko, and R. J. Mooney. A probabilistic framework for semi-supervised clustering. In *Proc. KDD*, 2004.

[3] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society B*, 39:1–38, 1977.

[4] P. Domingos and G. Hulten. Mining high-speed data streams. In *Proc. SIGKDD*, pages 71–80, 2000.

[5] J. Gehrke, V. Ganti, R. Ramakrishnan, and W. Loh. Boat-optimistic decision tree construction. In *Proc. SIGMOD*, 1999.

[6] M. M. Masud, J. Gao, L. Khan, J. Han, and B. Thuraisingham. A practical approach to classify evolving data streams: Training with limited amount of labeled data. *Univ. of Texas at Dallas Tech. Report# UTDCS-32-08 (http://www.utdallas.edu/˜mmm058000/reports/UTDCS-32-08.pdf)*, October 2008.

[7] M. Scholz and R. Klinkenberg. An ensemble classifier for drifting concepts. In *Proc. ICML/PKDD Workshop in Knowledge Discovery in Data Streams.*, 2005.

[8] K. Tumer and J. Ghosh. Error correlation and error reduction in ensemble classifiers. *Connection Science*, 8(304):385–403, 1996.

[9] K. Wagstaff, C. Cardie, S. Rogers, and S. Schroedl. Constrained k-means clustering with background knowledge. In *Proc. ICML*, pages 577–584, 2001.

[10] H. Wang, W. Fan, P. Yu, and J. Han. Mining concept-drifting data streams using ensemble classifiers. In *Proc. KDD*, 2003.